

# JS Level 1



# Цели курса



# Цели

Мы (команда Alif Skills) разработали линейку практикумов. В этих практикумах мы делимся с вами нашим опытом и предоставляем вам тренажёры, на которых вы сможете на практике отработать полученные навыки.



# Ключевое

**Ключевое:** для нас важно только то, насколько вы "прокачаетесь" (станете лучше, чем были до курса). Для нас совершенно не важно, за сколько времени вы сделали ДЗ, с какой попытки вы его сдали и т.д.

В реальной разработке важно только то, что вы уложились в сроки (дедлайн) и выполнили все требования. В "переносе на курс": важно только то, что вы сдали ДЗ в отведённые сроки.



# Цели

Цель практикумов JS – позволить вам стать FullStack-разработчиком.

Кто такой FullStack-разработчик? Это специалист, обладающий навыками в различных областях: от создания удобной пользовательской части (интерфейса), до разработки серверной части, взаимодействующей с базой данных и т.д.

Не пугайтесь непонятных слов, ключевое: вы станете специалистом, который способен с нуля самостоятельно разработать веб-приложение и развернуть его на сервере в сети Интернет.



# FullStack

Почему важно уметь самому делать полноценные веб-приложения?

1. Обладая такими навыками, вы будете понимать, как "всё устроено" и уметь изменять приложение самостоятельно (а не ждать, пока найдётся нужный специалист)
2. В современном мире очень важна скорость – вы можете придумать гениальную идею, но если не сможете её быстро реализовать, то она так и останется идеей – её вполне может реализовать кто-то другой

Поэтому в мы будем рассматривать только то, что имеет практическое значение и позволит вам научиться создавать реальные веб-приложения.



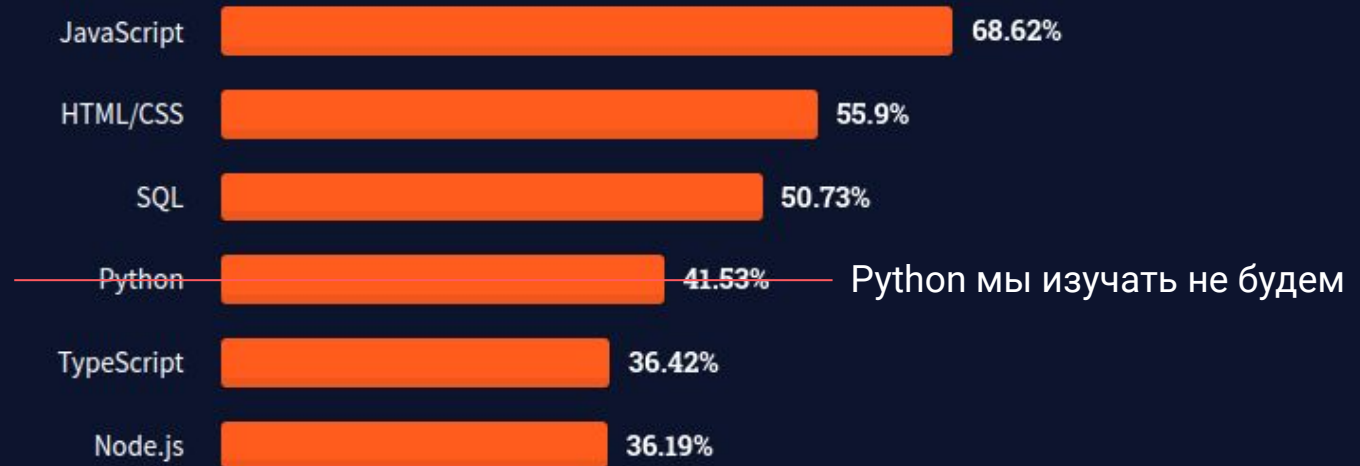
# Web

Для этого нам потребуется изучить не только JavaScript, но и сопутствующие технологии:

All Respondents

Professional Developers

58,031 responses



[Исследование StackOverflow](#) – самые популярные технологии 2021.



# Web

В рамках данного практикума мы научимся использовать JavaScript в среде веб-браузера (в том числе взаимодействовать с HTML и CSS).





# Введение



# Введение

Первые две лекции будут посвящены краткому повторению материала Level 0 (в той части, которая нужна для изучения JS), поэтому:

- если вы прошли Level 0 – для вас они не должны быть сложными
- а если не проходили – то как раз рассмотрите необходимый материал



# Инструменты



# Инструменты

Для прохождения основной части курса вам понадобятся три инструмента:

1. Браузер Chrome (желательно последней версии)
2. Платформа Node.js
3. Редактор VS Code (для дополнительных лекций понадобятся также другие инструменты).

В любом случае, если у вас возникнут проблемы, [пишите в чат группы](#), мы (и другие участники) вам поможем.



# Инструменты

Кроме того, важны следующие три момента:

1. У вас должны быть права администратора на компьютере (чтобы вы могли устанавливать программы)
2. Ваш пользователь должен называться по-английски, без пробелов в имени (если это не так – переименуйте)
3. Создавайте все проекты где-нибудь на диске C: (если у вас Windows), например, в каталоге `projects` (следите за тем, чтобы в именах каталогов и файлов не было пробелов, не английских символов и т.д.)



# Node.js

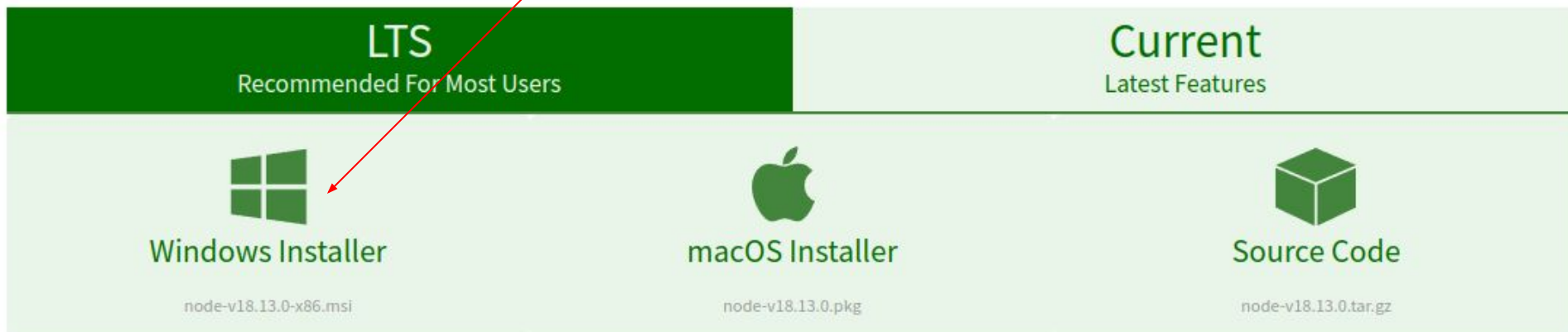
[Node.js](#) – это специальная среда для запуска JS приложений, позволяющая запускаться им вне браузера.

На Node.js написано большинство инструментов, которые используются Frontend-разработчиками. Кроме того, если мы захотим разрабатывать Backend (серверную часть приложения) на JS – Node.js также позволит нам это сделать.



# Node.js

Для установки Node.js, перейдите по адресу <https://nodejs.org/en/download/> и выберите установочный файл для вашей операционной системы (далее – ОС). Например, для Windows, нужно выбрать Windows Installer:



# Node.js

Если по каким-то причинам файлы не скачиваются, вы можете использовать следующие ссылки:

- <https://alif-skills.pro/media/node-v18.13.0-x86.msi> (32 битная версия)
- <https://alif-skills.pro/media/node-v18.13.0-x64.msi> (64 битная версия)





# Вирусы

Не забывайте проверять любые скачиваемые файлы с помощью сервиса [VirusTotal](https://www.virustotal.com):




Analyze suspicious files and URLs to detect types of malware, automatically  
share them with the security community

FILE

URL

SEARCH

A square icon with a document symbol and a fingerprint, representing file upload or analysis.

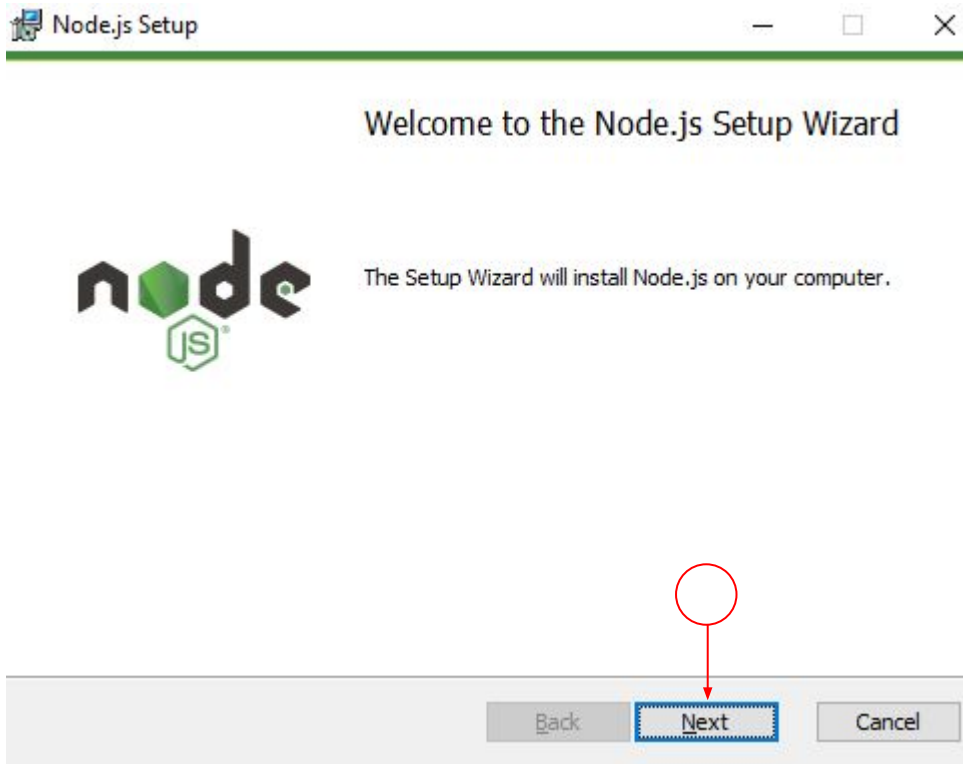
By submitting data below, you are agreeing to our [Terms of Service](#) and [Privacy Policy](#), and to the sharing of your Sample submission with the security community. Please do not submit any personal information; VirusTotal is not responsible for the contents of your submission. [Learn more.](#)

Choose file



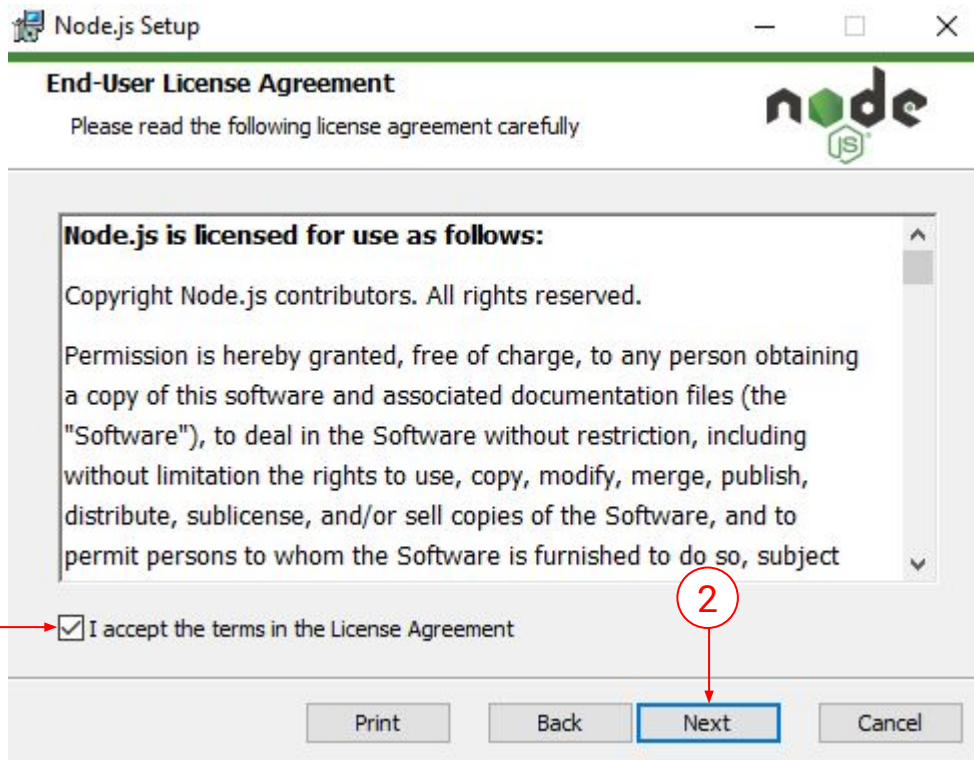
# Установка Node.js

Дождитесь активации кнопки **Next** (это может занять несколько минут), после чего нажмите на кнопку **Next**:



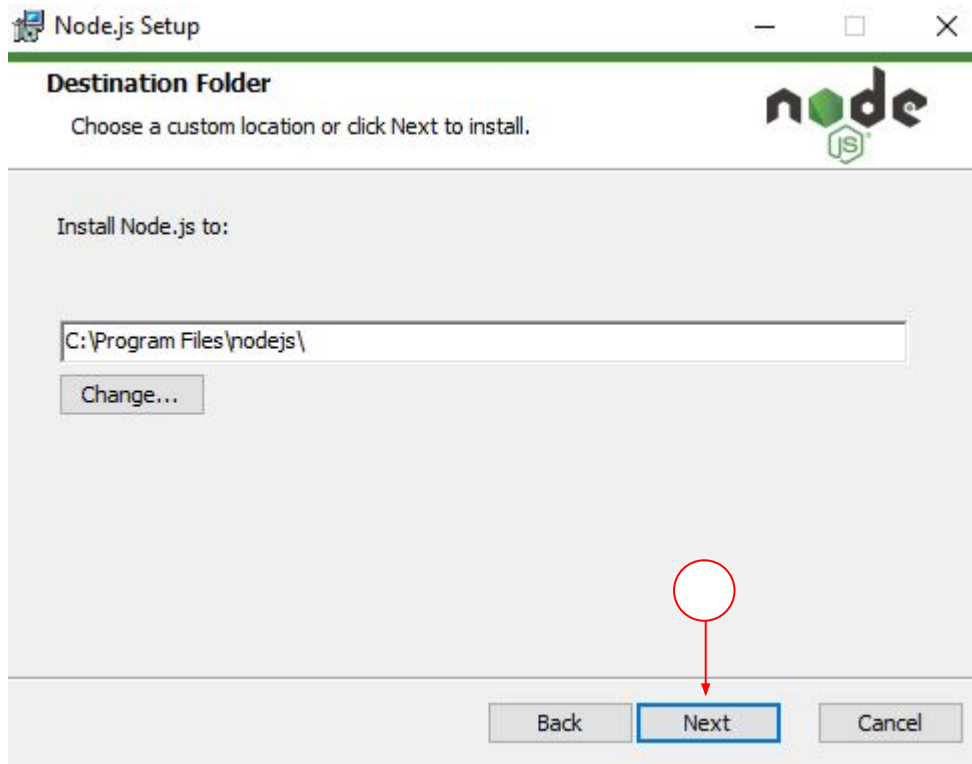
# Установка Node.js

Прочитайте и примите условия лицензионного соглашения, после чего нажмите на кнопку **Next**:



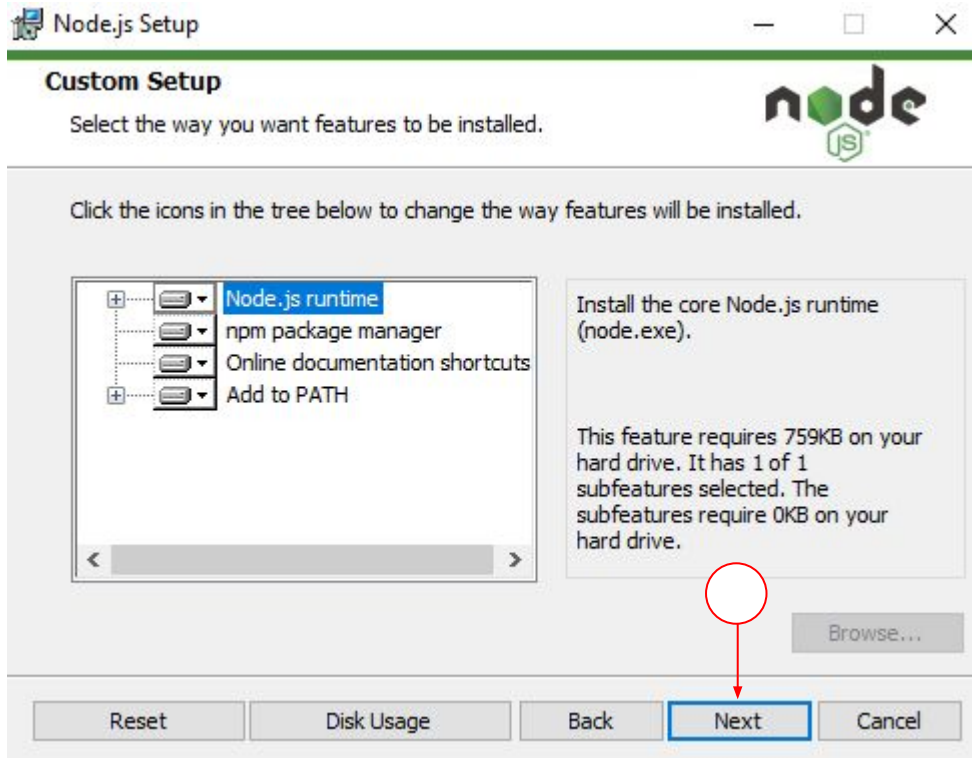
# Установка Node.js

Подтвердите установку в указанный каталог (нужно нажать кнопку **Next**):



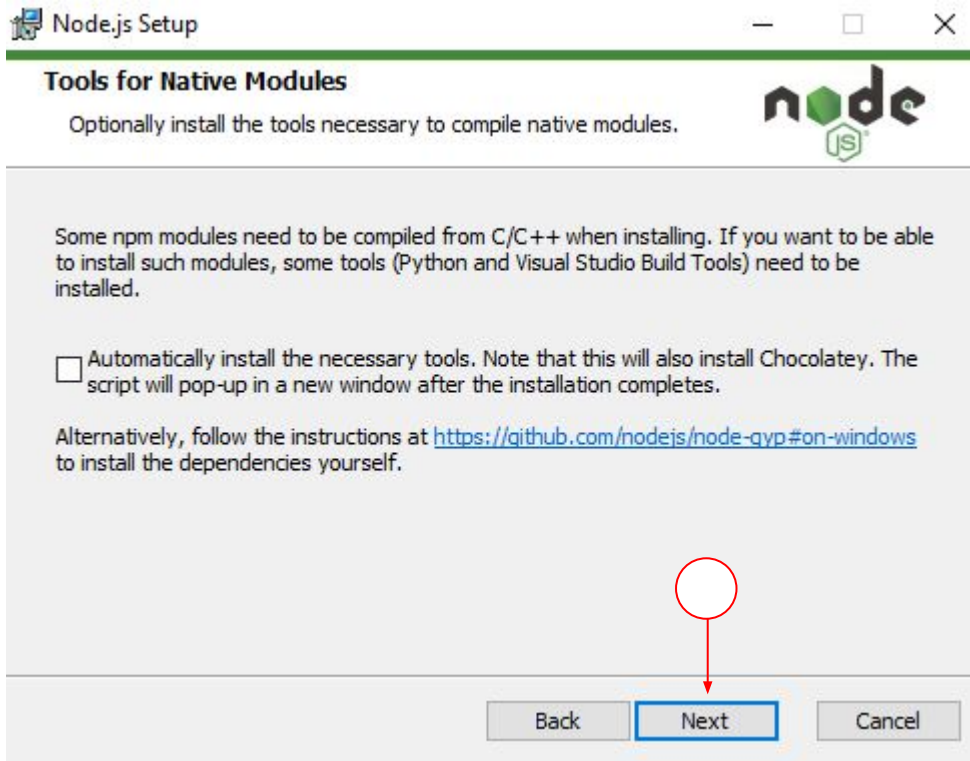
# Установка Node.js

Оставьте выбранные значения по умолчанию и нажмите на кнопку **Next**:



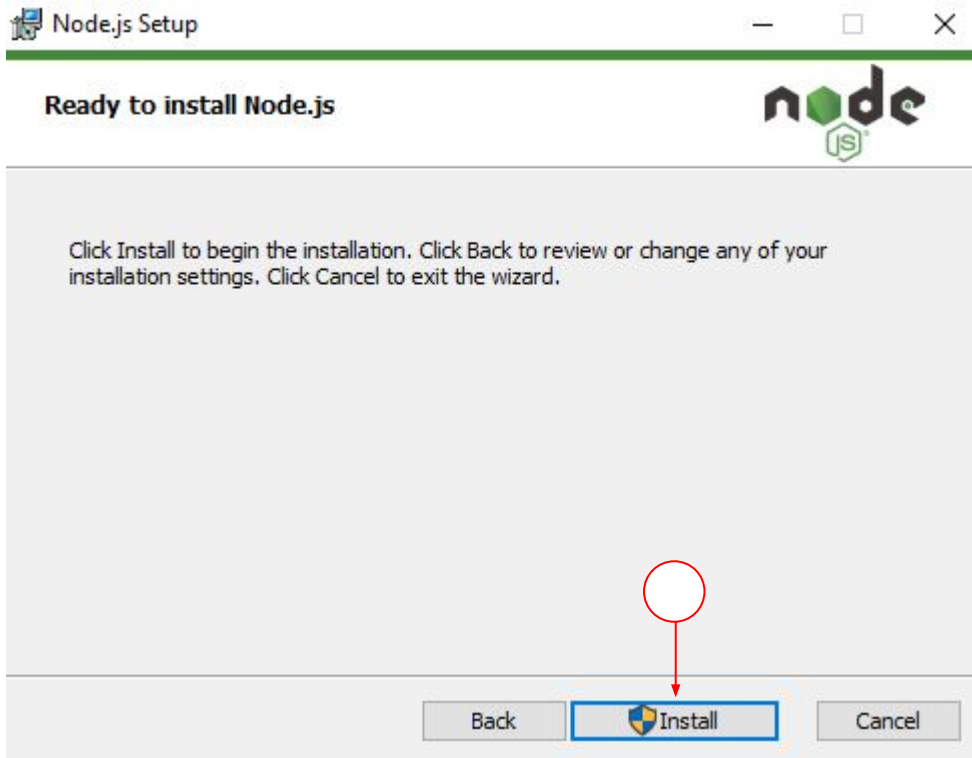
# Установка Node.js

Оставьте выбранные значения по умолчанию и нажмите на кнопку **Next**:



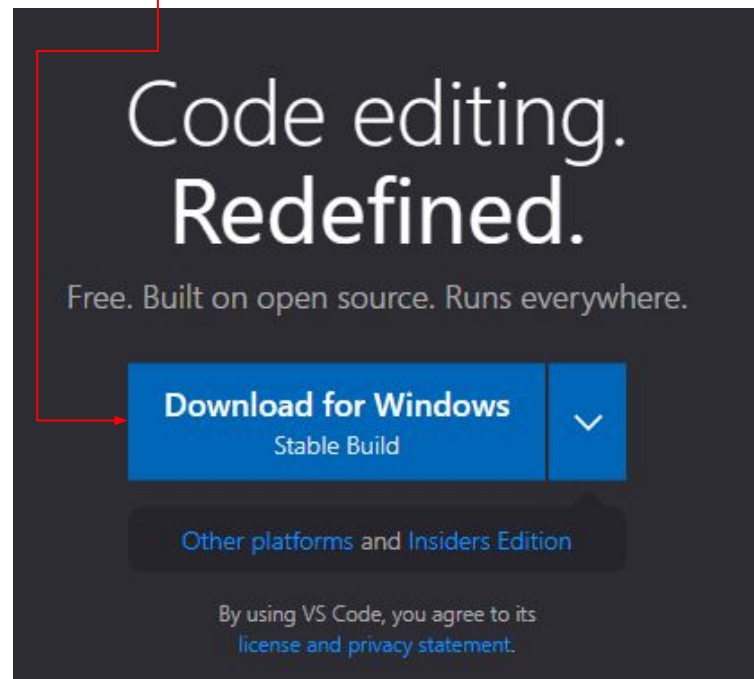
# Установка Node.js

Подтвердите установку нажатием на кнопку **Install**:



# VS Code

VS Code самый популярный редактор кода (специальная программа, которая позволяет вам писать код). Перейдите по адресу [code.visualstudio.com](https://code.visualstudio.com) и скачайте установочный файл для вашей операционной системы. Например, для Windows, нужно выбрать [Download For Windows](#) (если не Windows, то выберите свою ОС из выпадающего списка):





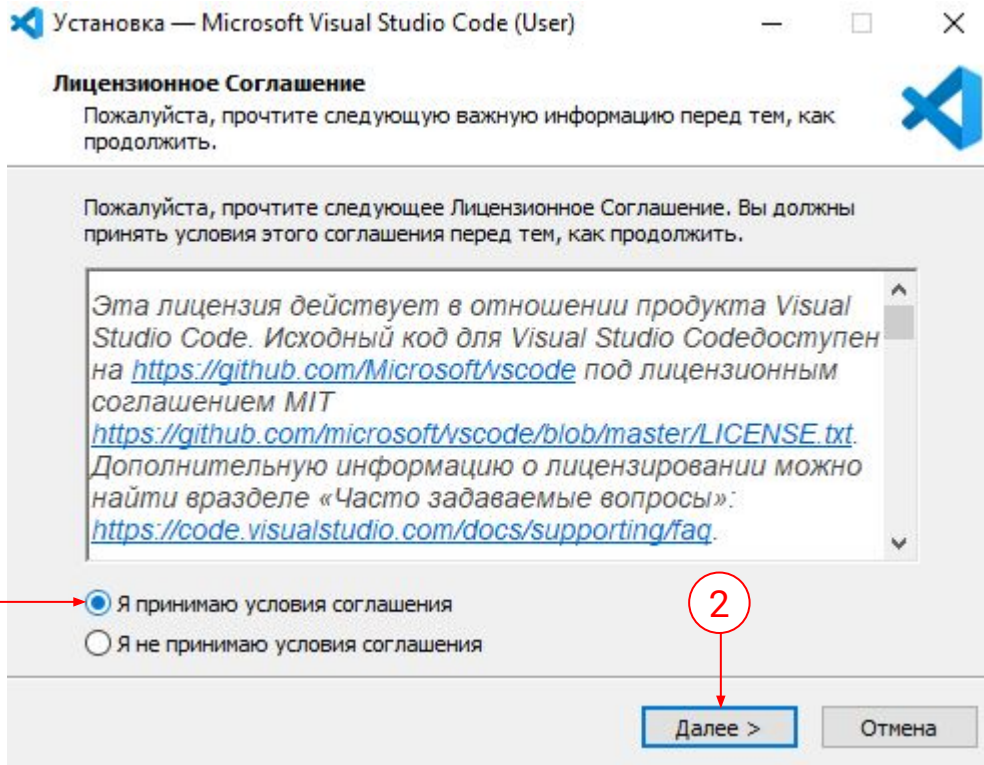
# VS Code

Если с официального сайта скачать не получилось, вы можете скачать файл по адресу: <https://alif-skills.pro/media/vscode.exe>.



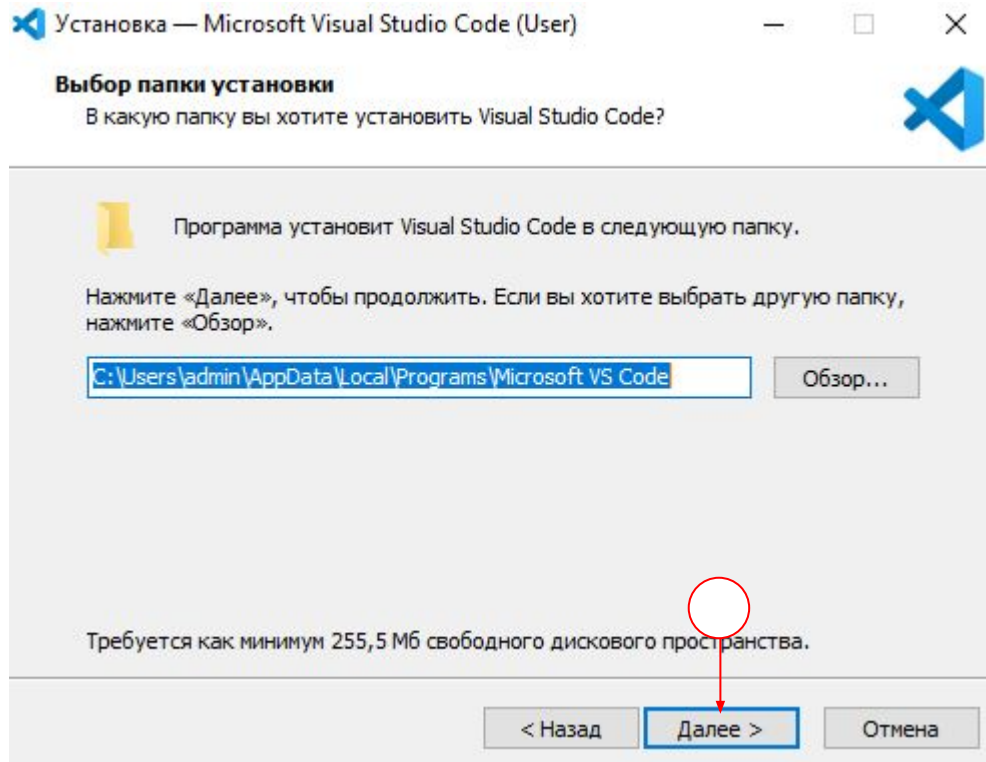
# VS Code

Прочитайте и примите условия лицензионного соглашения, после чего нажмите на кнопку **Далее**:



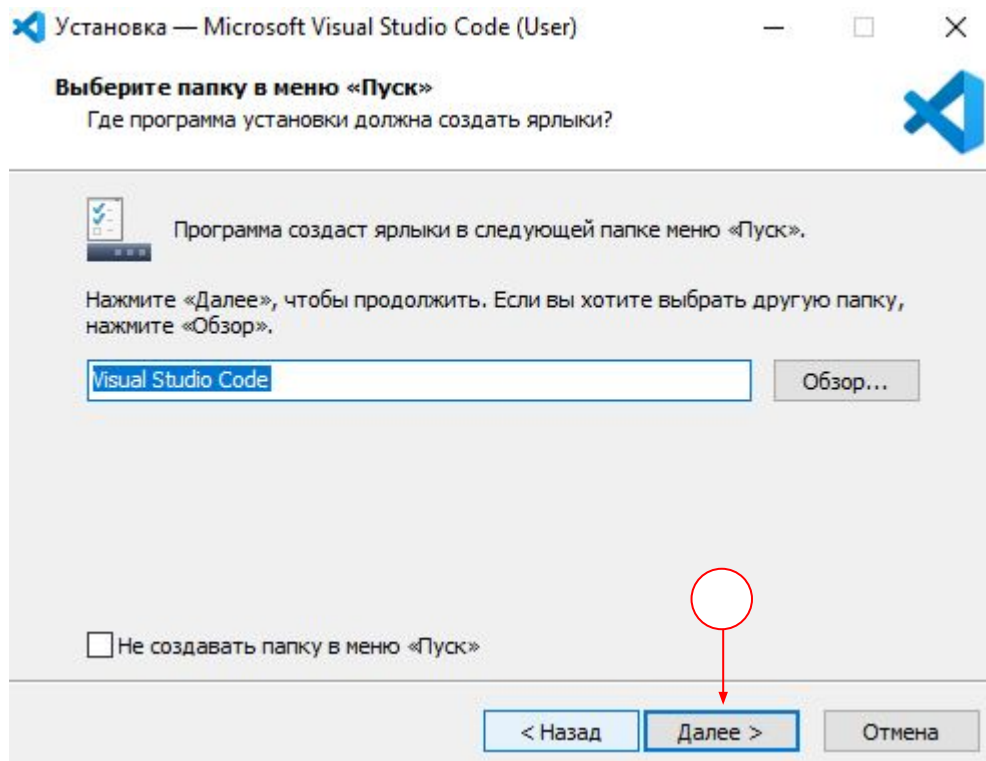
# VS Code

Оставьте значения по умолчанию, после чего нажмите на кнопку **Далее**:



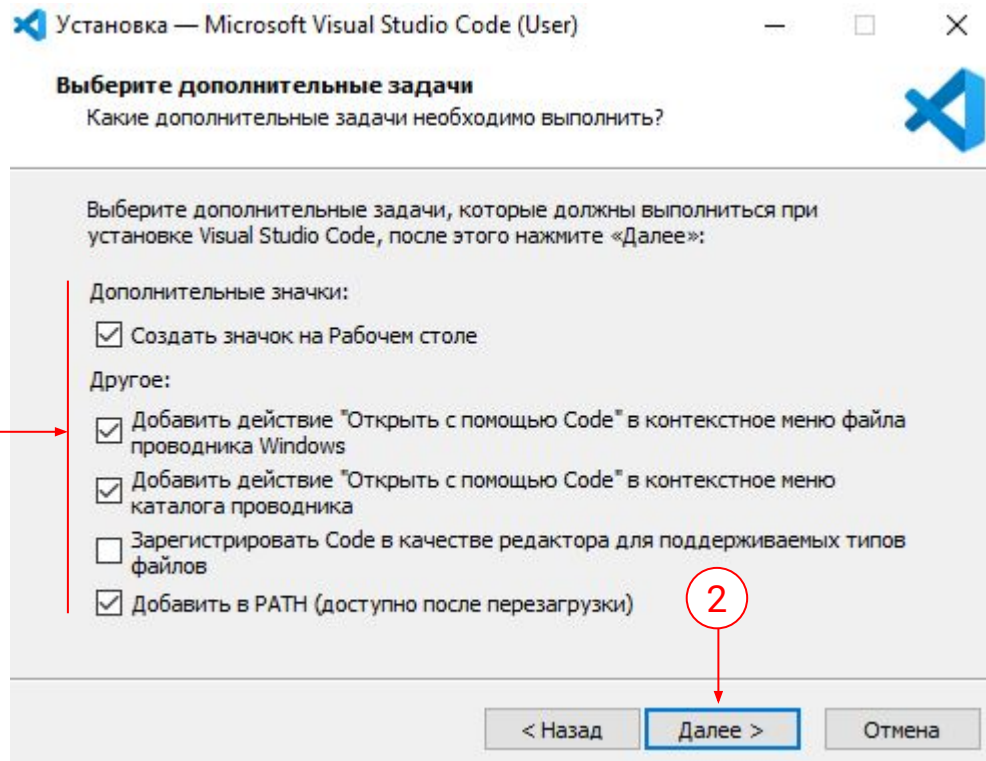
# VS Code

Оставьте значения по умолчанию, после чего нажмите на кнопку **Далее**:



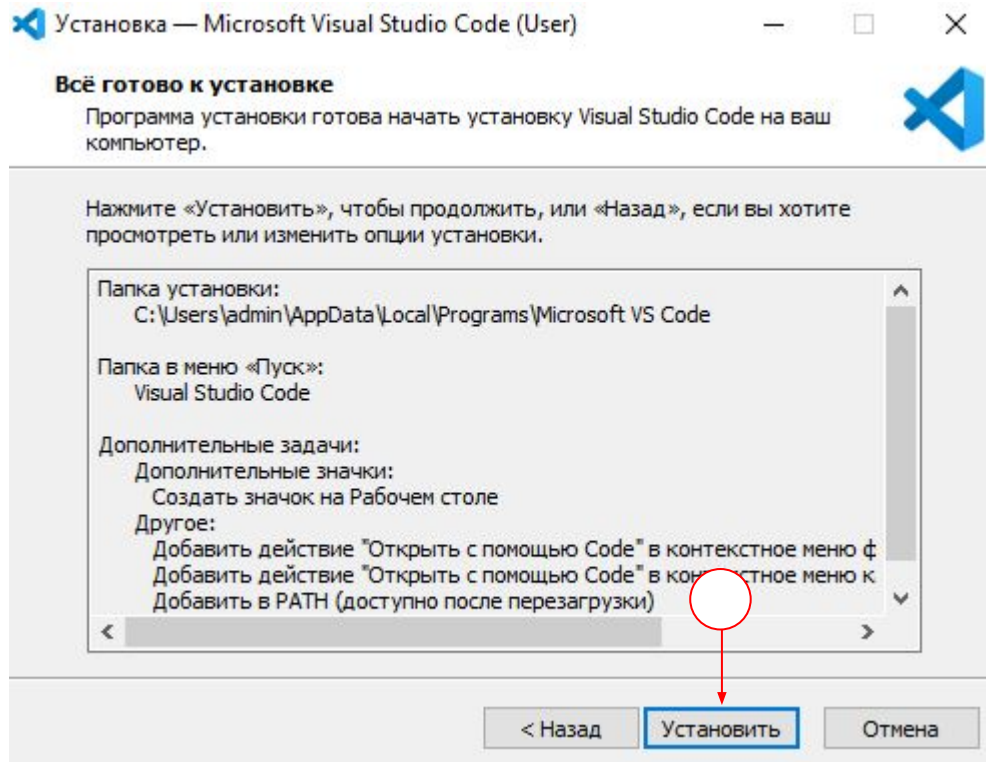
# VS Code

Поставьте флажки как на скриншоте, после чего нажмите на кнопку **Далее**:



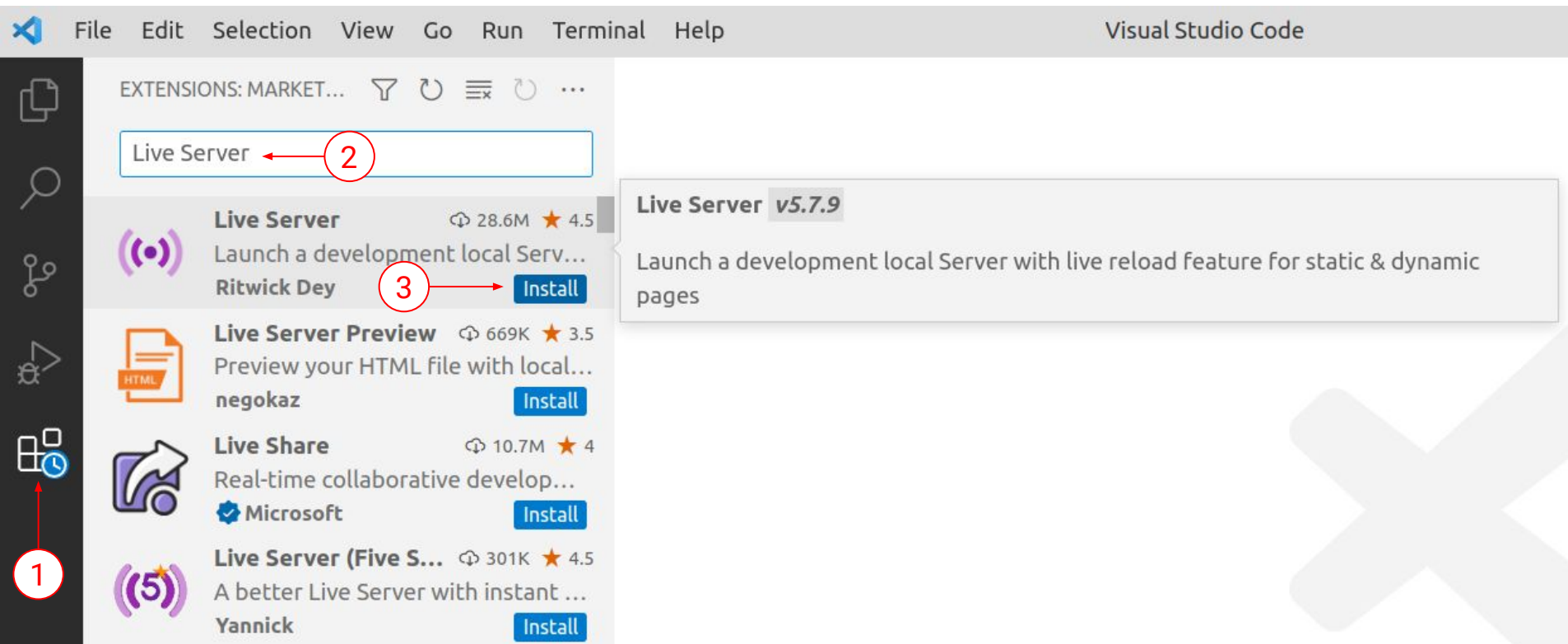
# VS Code

Подтвердите установку, нажав на кнопку **Установить**:



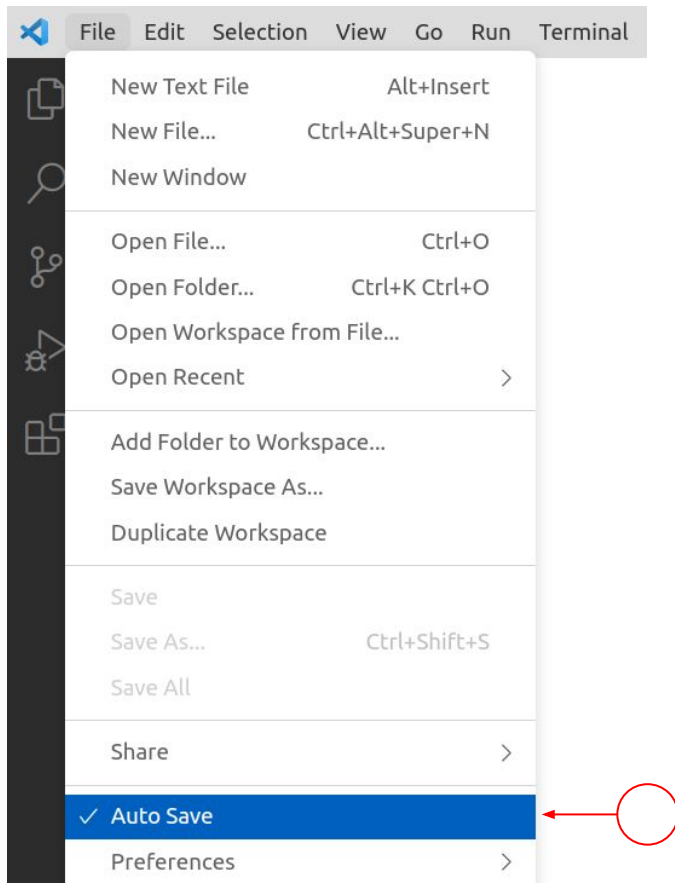
# VS Code Live Server

После завершения установки запустите VS Code, зайдите в панельку [Extensions](#) (1),  
наберите [Live Server](#) (2) и нажмите на кнопку [Install](#) (3):



# VS Code Auto Save

Чтобы изменения, которые вы вносите в файлы автоматически сохранялись, зайдите в пункт меню **File** и выберите **Auto Save**:





# Web



# Web

Веб-приложение – это клиент-серверное приложение, в которой клиент – это приложение, работающее в веб-браузере.

Давайте разбираться по порядку. Что такое клиент-серверное? Это значит, что само приложение разделено на две части: есть клиент и есть сервер.



# Web

В веб-приложениях, клиент – это приложение, работающее в браузере (а браузер работает на вашем компьютере или смартфоне), а сервер (чаще всего) – это приложение, которое расположено на каком-то компьютере, подключенном к сети Интернет.

Эти две части приложения взаимодействуют друг с другом, обмениваясь информацией.



# Mobile Application

Нам будет проще понять, если мы будем воспринимать это так же, как мобильные приложения: например, на нашем смартфоне установлен Telegram – для этого мы его скачали и установили. Но чтобы полноценно им пользоваться, нам нужен ещё Интернет – без Интернета мы не сможем отправлять и получать сообщения:

Приложение внутри смартфона



# Web Application

С веб-приложениями всё почти так же, но чуточку сложнее: на нашем компьютере или смартфоне установлен браузер, а уже внутри браузера работает приложение:

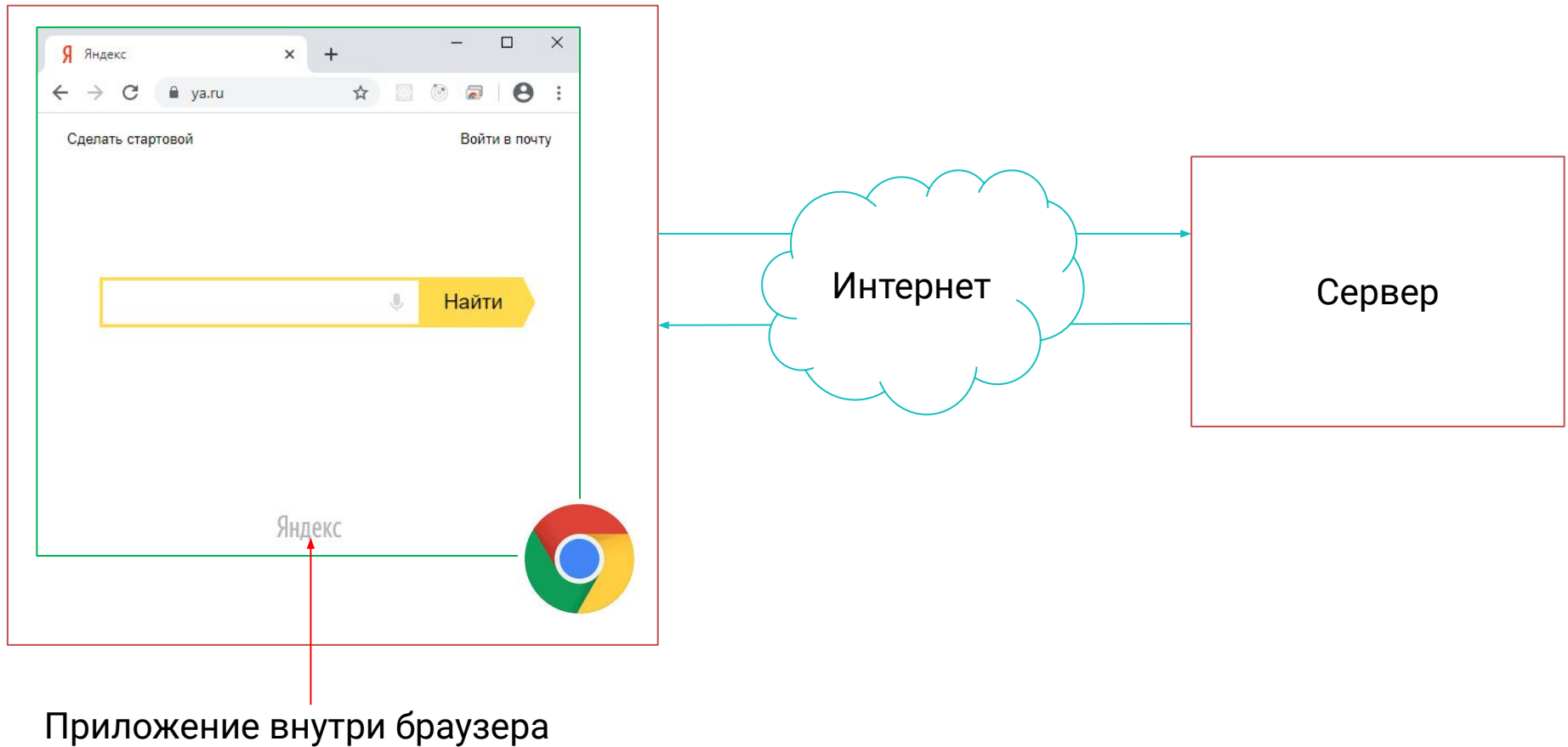


Например, если мы открываем страницу [www.ya.ru](http://www.ya.ru), то браузер загружает и запускает клиентскую часть приложения (в мобильных приложениях мы делаем это руками\* через Google Play или AppStore).

Примечание\*: не всегда это необходимо, существует Instant App и аналоги.



# Web Application



# Упрощения

Нас пока не интересует, что из себя представляет сервер, как браузер загружает и запускает наше приложение и т.д.

Ключевое: мы должны понять саму аналогию – браузер внутри себя запускает программу.

Итого у нас две части одного большого приложения:

1. Внутри браузера (клиент)
2. На сервере (сервер)

Привыкните к тому, что вы не сможете сразу понять как всё устроено: сервер, клиент, передача данных и т.д. Вместо этого учитесь улавливать ключевые идеи и применять их сразу на практике.



# Определения

1. **Web (World Wide Web)\*** – система связанных веб-страниц, доступных через Интернет. Что значит "связанных"? Мы знаем, что можно переходить от одной странице к другой, используя ссылки (или гиперссылки). Это и есть связность страниц
2. **Веб-страница** – документ, который может быть отображён веб-браузером. Что значит документ? Для простоты будем считать, что документ – это просто файл с расширением [.html](#)
3. **Браузер (веб-браузер)** – программа для просмотра веб-страниц и перехода между ними посредством гиперссылок. Именно браузер, как мы уже с вами говорили, отвечает за то, чтобы запускать клиентскую часть приложения

Примечание\*: определения, по большей части, взяты с сайта

<https://developer.mozilla.org>





# HTML, CSS, JS

Теперь нам нужно разобраться, как именно делать то, что может запускать браузер. Для этого у нас есть набор технологий:

- **HTML** – позволяет создавать документы, которые умеет обрабатывать браузер\* (те самые веб-страницы)
- **CSS** – позволяет визуально оформлять документы
- **JS** – позволяет писать программы, взаимодействующие с документом (и не только)

Давайте разбираться с каждой из них.

Примечание\*: строго говоря, браузер умеет обрабатывать огромное количество различных типов файлов (картинки, видео, pdf), но под термином документ (если не указано другое) мы будем понимать именно HTML.



# HTML

**HyperText Markup Language (HTML)** – язык разметки, определяющий структуру документа.

Очень сложное и непонятное определение. Что оно на самом деле значит? "Язык" означает, что существуют определённые правила, которым нужно следовать, чтобы вас поняли. Т.е. мы соблюдаем правила – браузер делает то, что нам нужно, не соблюдаем – никто за результат не отвечает (что нас, естественно, не устраивает).



# HTML

Ключевые моменты:

1. Если есть правила, они должны быть где-то описаны. В случае HTML – это [спецификация](#) (или стандарт)
2. Если мы получаем не тот результат, который ожидаем, то виноваты только мы, а не браузер\*

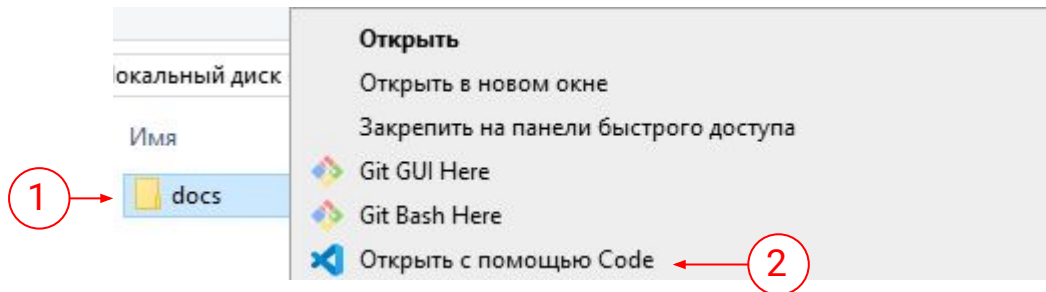
Примечание\*: конечно, иногда в браузерах есть ошибки, но чаще будем виноваты именно мы, а не ошибка в браузере.



# HTML

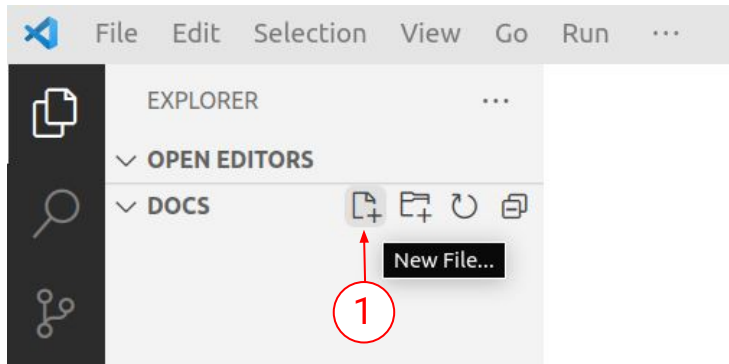
Мы с вами будем детально разбираться HTML и CSS в рамках этого курса.

Создадим в каталоге наших проектов каталог **docs** и откроем его в VS Code (клик правой кнопкой мыши на каталоге):

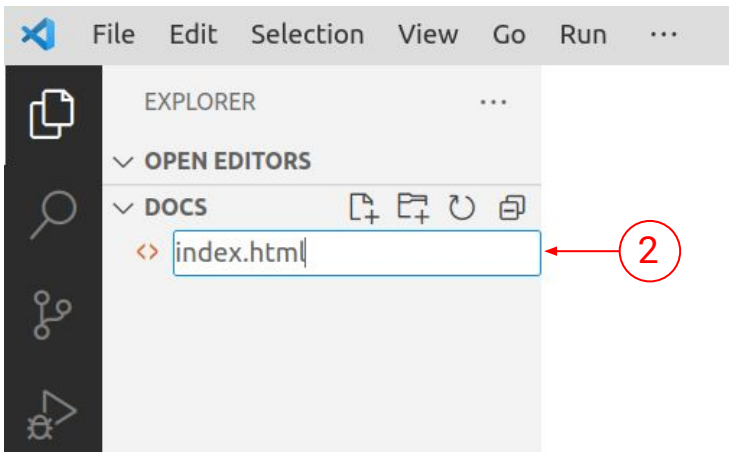


# HTML

В боковой панельке нажмите на **+** для создания нового файла:

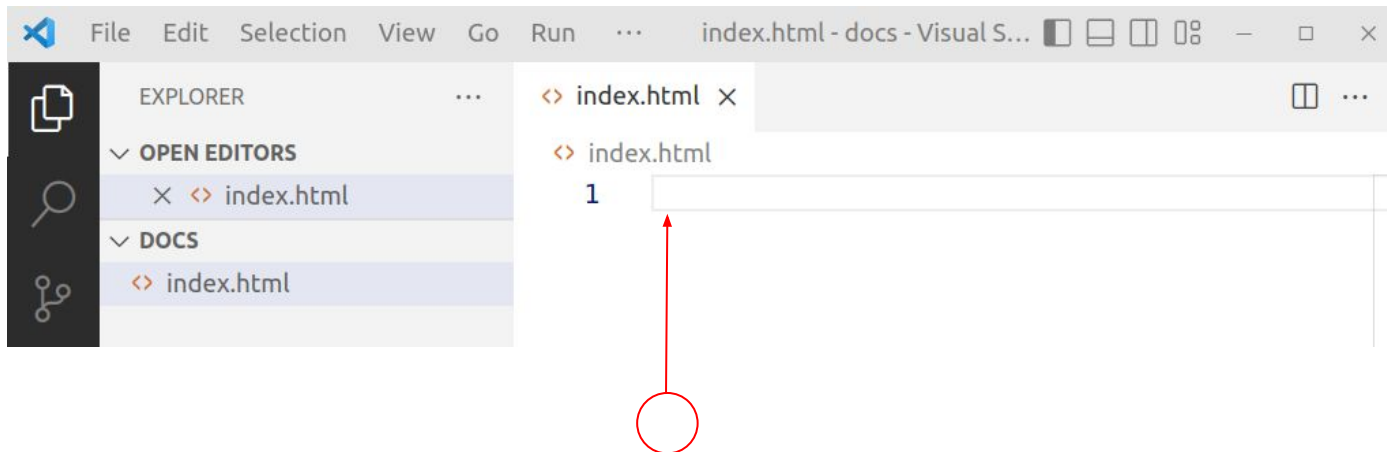


Введите **index.html** и нажмите на **Enter**:



# HTML

После этого файл откроется для редактирования в основной части редактора:



Там мы и будем писать код.



# index.html

`index.html` – общепринятое имя для основного файла вашего веб-приложения.

Несмотря на то, что можно использовать и другое имя, считается хорошим правилом следовать идее "Convention over Configuration" (следуем общепринятым соглашениям вместо того, чтобы что-то настраивать).



# HTML Document





# HTML Document

Базовая структура HTML-документа выглядит следующим образом (пока этот код набирать не нужно):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello, Alif Skills!</h1>
</body>
</html>
```

Давайте разберёмся с основными правилами.



# HTML Document

Весь документ состоит из HTML элементов и некоторых других сущностей. У нас из таких "специальных" сущностей есть только `<!DOCTYPE html>` (произносится как "доктайп"), с которой и начинается наш документ.

`<!DOCTYPE html>` – это специальная конструкция, которая сообщает браузеру, что этот документ стоит обрабатывать как HTML самой последней версии.

Весь остальной документ состоит из набора HTML элементов.



# HTML Element

HTML элемент представляет из себя специальную запись:



Пишется всё именно в таком порядке: сначала открывающий (`<h1>`), затем содержимое (в данном случае – текст `"Hello, Alif Skills!"`) и только потом закрывающий (`</h1>`).

Закрывающий тег от открывающего отличается наличием символа `/`.



# HTML Element

Все допустимые элементы (те, которые понимает браузер) описаны в [спецификации](#). Элементы – это те "кирпичики", из которых строится страница.

Например, есть элементы для отображения картинок, аудио, видео. Элемент **h1** предназначен для отображения заголовка первого уровня (самого главного заголовка на странице). Помимо него есть ещё **h2**, **h3**, **h4**, **h5**, **h6** (представьте, что мы пишем реферат, курсовую или диплом – у нас есть название всего документа, название раздела, подраздела и т.д., в HTML документе всё так же).

**Важно:** мы пока не разбираем, какие есть элементы и для чего они нужны (всё будем делать по-порядку). Сейчас наша задача – понять общие правила того, как всё устроено.



# Вложенность

Элементы можно вкладывать друг в друга, т.е. внутрь элементов можно помещать текст и другие элементы:

```
<body>  
| <h1>Hello, Alif Skills!</h1>  
</body>
```

Здесь внутри элемента `body` расположен элемент `h1`. Это позволяет организовывать дерево элементов: тот элемент, в который вкладывают другие, называется **родительским элементом** (`body` родительский элемент для `h1`), а те, которые вкладывают, называют **дочерними элементами** (`h1` дочерний элемент для `body`).

**Важно правило:** у каждого элемента есть только один родительский элемент.



# Вложенность

Очень важно соблюдать правило вложенности – элемент не может "выходить" за границы родительского элемента.

Пример правильной вложенности:

```
<body>  
| <h1>Hello, <span>Alif Skills</span>!</h1>  
</body>
```

Элемент не выходит за "границы" родительского

Примеры неправильной вложенности (внимательно следите за вложенностью!):

```
<body>  
| <h1>Hello, <span>Alif Skills!</h1></span>  
</body>
```

Элемент **выходит** за "границы" родительского

```
<body>  
| <h1>Hello, <span>Alif Skills!</h1>  
</body>
```

Отсутствует закрывающий тег



# Q & A

**Q:** что делает браузер, когда встречает неправильно оформленный документ (например, не соблюдено правило вложенности)?

**A:** он исправляет документ так, как считает нужным. **НО:** он может исправить его не так, как нужно нам! Поэтому важно сразу писать правильно оформленные документы.



# Закрывающий тег

Для некоторых элементов работает следующее правило: если у элемента нет содержимого (текста внутри или вложенных элементов), то можно использовать сокращённую запись:

```
<meta charset="UTF-8"/>
```

Либо вообще опускать:

```
<meta charset="UTF-8">
```

Мы разберём чуть позже для каких элементов это правило работает.





# Атрибуты

Элемент может содержать атрибуты (располагаются только в открывающем теге):

`<meta charset="UTF-8">`

имя  
атрибута

значение

Атрибуты позволяют "настраивать" свойства элемента. Например, через атрибуты можно будет задать "громкость" аудио или видео-элемента, ширину, высоту и т.д.



# Emmet

Мы с вами разобрали ключевые моменты приведённого фрагмента HTML-кода.

Самый главный вопрос: "Как эту базовую структуру создать"? Ведь запомнить её с ходу достаточно сложно.



# Emmet

На самом деле, запоминать её и не нужно, потому что вам будет помогать специальный инструмент [Emmet](#) (который уже встроен в VS Code). Наберите в редакторе символ **!** и подождите, пока появится подсказка, после чего нажмите клавишу **Tab**:



Если вдруг подсказка пропала или не появляется, нажмите на клавиатуре **Ctrl + пробел** (плюс набирать не надо, это значит, что нужно одновременно нажать **Ctrl** и **пробел**). Скрыть подсказку можно с помощью клавиши **Esc**.



# Emmet

Emmet за нас напишет типовую структуру:

<> index.html > html > head > meta

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Document</title>
8  </head>
9  <body>
10  |
11 </body>
12 </html>
```

у вас может не быть этой строки (это не критично)

Нажимая на клавишу **Tab** мы можем перемещаться между подсвеченными областями, пока не попадёте внутрь элемента **body**.



# Emmet: элементы

Помимо базовой структуры, мы можем с помощью Emmet создавать и элементы.

Для этого нужно написать название тега элемента и нажать на **Tab**:

```
<body>  
  h1  
</bo  h1 Emmet Abbreviation  
</html>
```

Нажатие на **Tab** создаст открывающий и закрывающий тег + поместит курсор между ними

```
<body>  
  <h1>|</h1>  
</body>  
</html>
```

Теперь внутри элемента мы можем написать **Hello, Alif Skills!**



# Emmet

Emmet имеет ещё кучу других возможностей, с которыми мы познакомимся в рамках курса.

Например, он знает, для каких элементов закрывающий тег не обязателен и не будет его писать (а вам не придётся это запоминать, если вы пользуетесь Emmet). Чтобы быстро освоить Emmet (как и любой другой инструмент), принято пользоваться [шпаргалками](https://alif-skills.pro/media/emmet.pdf) (cheatsheets). Если документ по ссылке не доступен, скачайте его с <https://alif-skills.pro/media/emmet.pdf>.

**Важно:** привыкайте с первых дней меньше писать и больше пользоваться инструментами. Так вы будете гораздо продуктивнее.



# Работа браузера



# Live Server

Мы с вами создали документ, теперь нам нужно научиться "загружать" его в браузер и смотреть на результат.

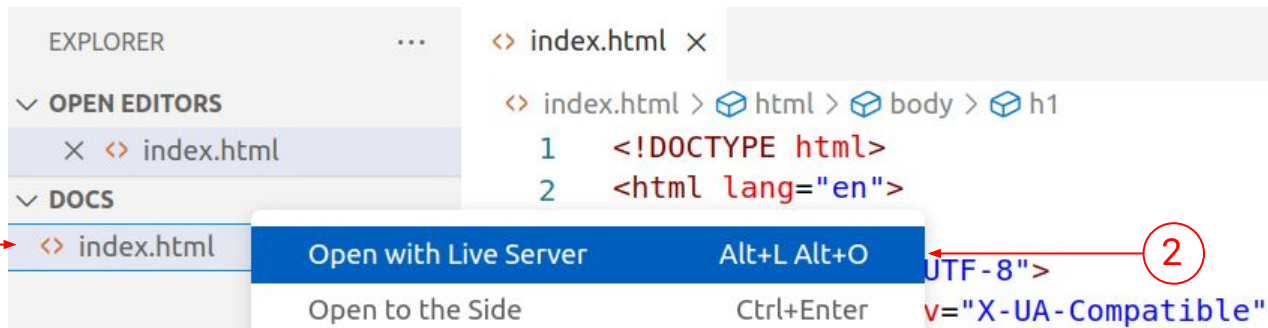
Можно, конечно, открыть его просто так (пойти в каталог с файлом и открыть двойным кликом), но это неправильный подход. Напоминаем, мы хотим добиться следующей схемы:





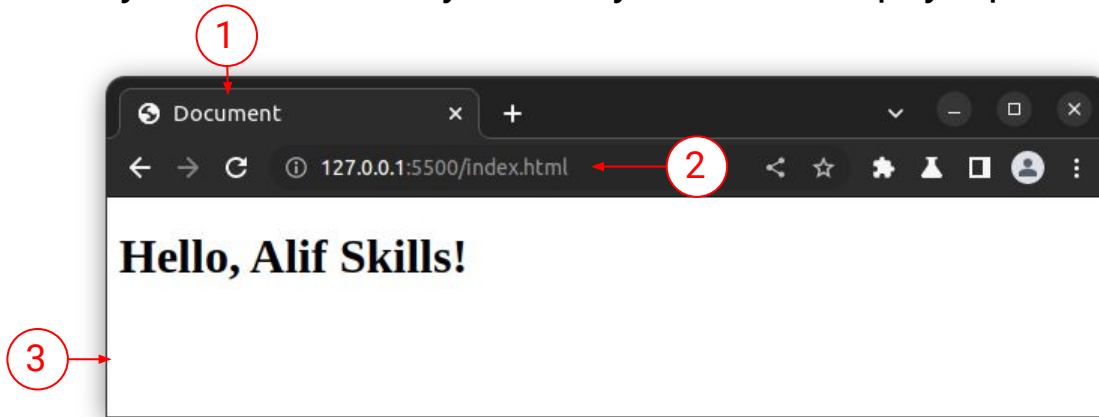
# Live Server

Щёлкните правой кнопкой мыши на `index.html` (1) и выберите `Open with Live Server` (2):



# Live Server

В установленном у вас по умолчанию браузере откроется следующая страница:



Ключевые вопросы:

1. Откуда взялось название вкладки (1)?
2. Что за адрес в адресной строке (2)?
3. Почему из всего кода, что мы написали, мы видим только содержимое **h1** (3)?

**Важно:** если у вас пустая белая страница, убедитесь, что вы написали внутри **h1** текст и сохранили страницу (включили авто-сохранение).



# HTML

В HTML документе есть две большие "части": **head** и **body**.

**head** отвечает за мета-данные страницы (т.е. специальные служебные данные для браузера) – например, название вкладки (**title**) или масштаб.

**body** же отвечает за те данные, которые будут отображены на самой странице, именно поэтому мы видим содержимое **h1**.

```
<> index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Document</title>
8  </head>
9  <body>
10   <h1>Hello, Alif Skills!</h1>
11 </body>
12 </html>
```

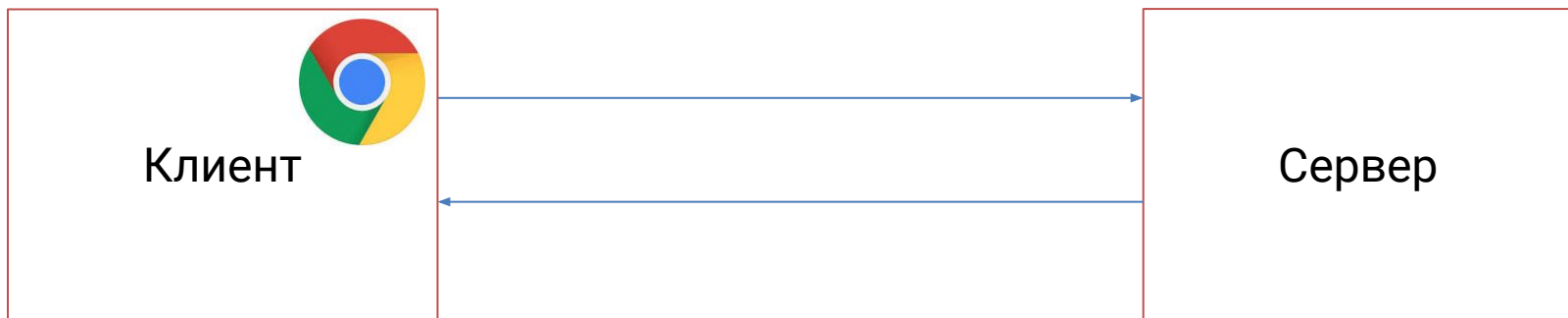


# Live Server

При установке VS Code, мы установили расширение Live Server, которое создаёт сервер прямо на нашем компьютере. И этот сервер работает по адресу:

<http://127.0.0.1:5500>.

При этом нам даже не нужен Интернет и общая схема нашей работы будет выглядеть вот так (всё работает внутри одного компьютера):



Live Server полезен ещё тем, что автоматически обновит страницу в браузере, если мы туда внесём изменения (попробуйте), т.е. мы сразу будем видеть все изменения.



# Live Server

У вас может возникнуть вопрос "Зачем нам Live Server, если мы и без него можем открыть страницу в браузере и обновлять?".

Причин несколько:

1. Обновлять руками – неудобно
2. Браузер запретит вам использовать некоторые возможности из JS в дальнейшем, если вы будете открывать HTML-документ просто двойным кликом в файловом менеджере

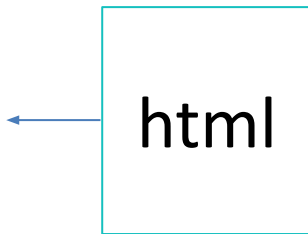
Поэтому обязательно используйте Live Server. Все лекции будут построены на том, что вы смотрите на результат работы именно так, а не открывая файл двойным кликом.



# Как работает браузер

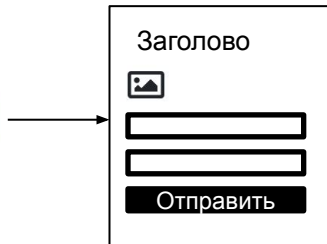
Теперь давайте разберёмся с тем, как работает браузер: браузер скачивает по адресу документ и обрабатывает его в соответствии со спецификацией, превращая документ в визуальное отображение для пользователя:

1



загрузка документа

2



визуальное отображение



# CSS

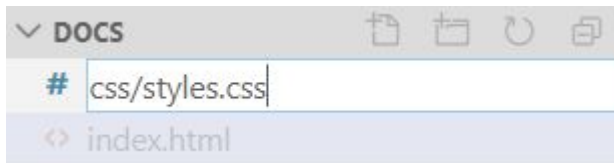


# CSS

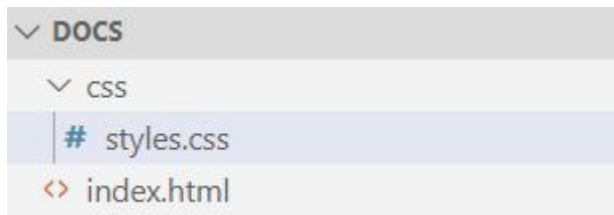
**CSS (Cascading Style Sheets)** – язык стилей, позволяющий определить визуальное представление структурированных документов (в частности, HTML-документов).

Давайте попробуем создать эти самые стили и применить их к нашему документу.

Кликните на **+** и введите `css/styles.css`:



Подобная запись позволяет нам создать каталог `css`, в котором будет храниться файл `styles.css` (в рамках этого курса мы css-файлы складывать в каталог `css`):





# CSS

В упрощённом варианте CSS представляет из себя набор правил вида:

селектор → `body {`  
`color: ■ grey;`  
`background-color: □ whitesmoke;`  
`}`

свойство                      значение

Селектор – это специальное выражение, которое позволяет выбирать элементы на странице. В текущем виде (`body`) будет выбран элемент `body` и к нему применены описанные стили: цвет текста (`color`) серый (`grey`), цвет фона (`background-color`) "белый дым" (`whitesmoke`).

Мы ещё будем разбираться с CSS, пока же нам достаточно того, чтобы этот код был в вашем файле.



# Ресурсы

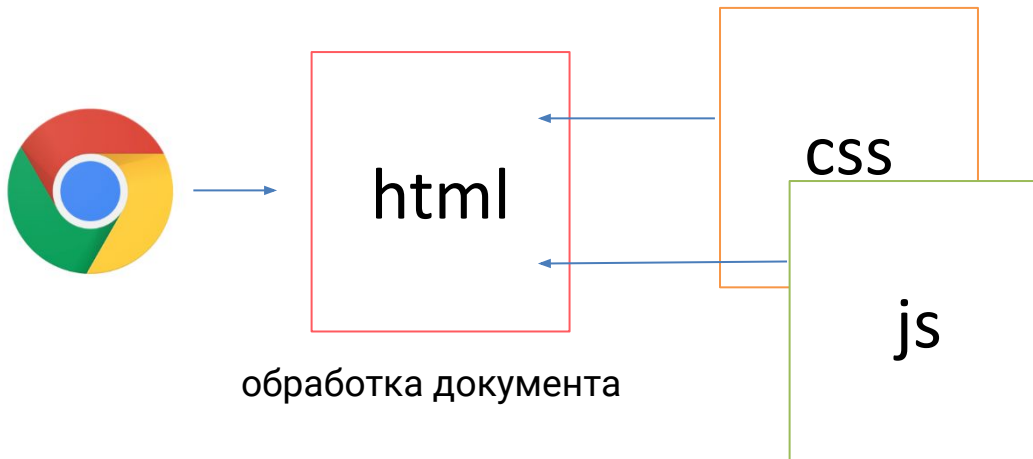
Если мы посмотрим на нашу страницу, то ничего не изменится (и даже если обновим её вручную, подумав, что Live Server не справился). По одной простой причине: браузер ничего не знает о наших стилях.

Это один из ключевых моментов – вы периодически будете сталкиваться с тем, что что-то не работает, по одной простой причине – вы забыли "подключить" ваши файлы.



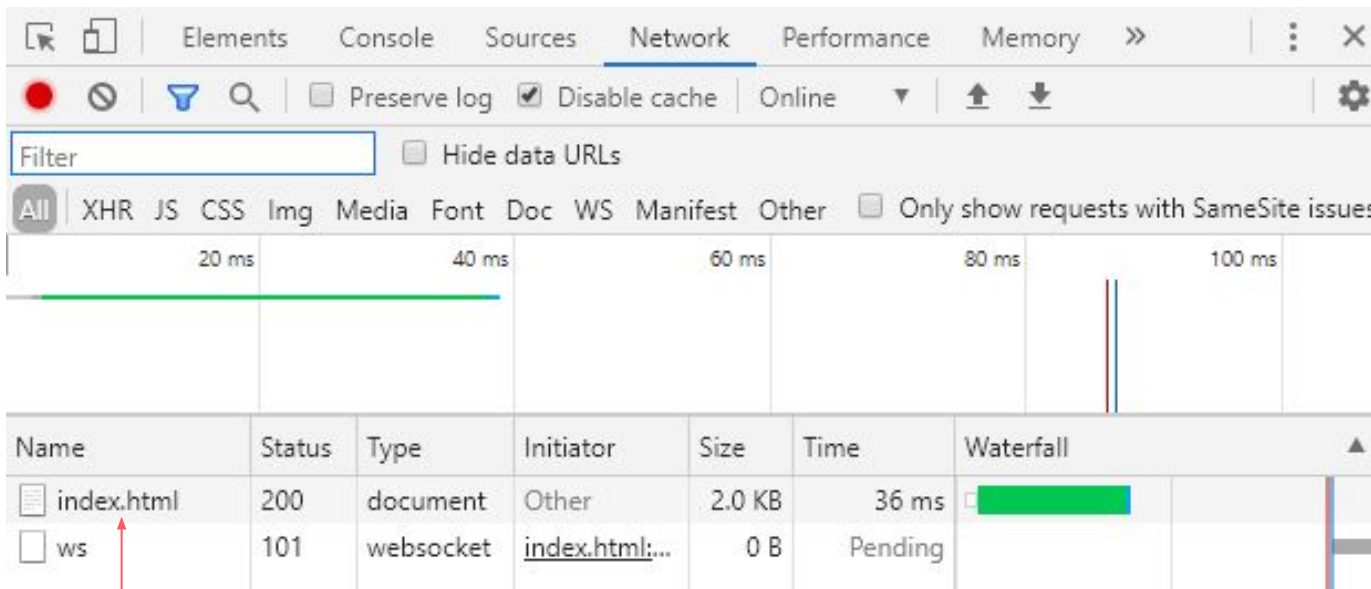
# Ресурсы

Как же их подключать? Мы с вами говорили, что браузер загружает HTML-документ и обрабатывает его. Если браузер встречает в HTML-документе указание на другие ресурсы (например, css-файлы или картинки), то он (браузер) загружает и обрабатывает и их в том числе:



# Developer Tools

Для того, чтобы проверить, какие файлы браузер загружает, необходимо в браузере нажать кнопку **F12** (если у вас ноутбук, то, возможно, **Fn + F12**) или **Ctrl + Shift + I** и перейти на вкладку **Network** (после чего обновить страницу – **F5** или кнопка обновить в браузере):



Developer Tools

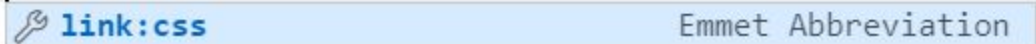
Как вы видите, здесь только **index.html** (**ws** – добавляет Live Server, мы его будем игнорировать).



# Ресурсы

Для того, чтобы подключить CSS-стили к нашему приложению, необходимо в элементе `head` прописать `link:css` и нажать `Tab` (ещё одно сокращение Emmet):

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  link:css|
</head>
```



Затем значение атрибута `href` заменить на `"css/styles.css"`:

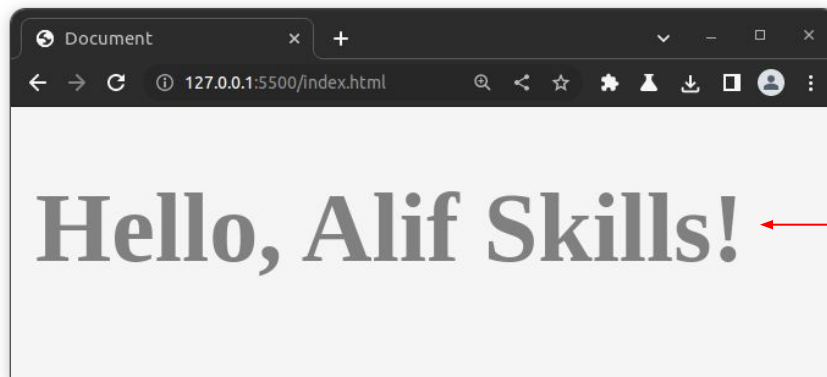
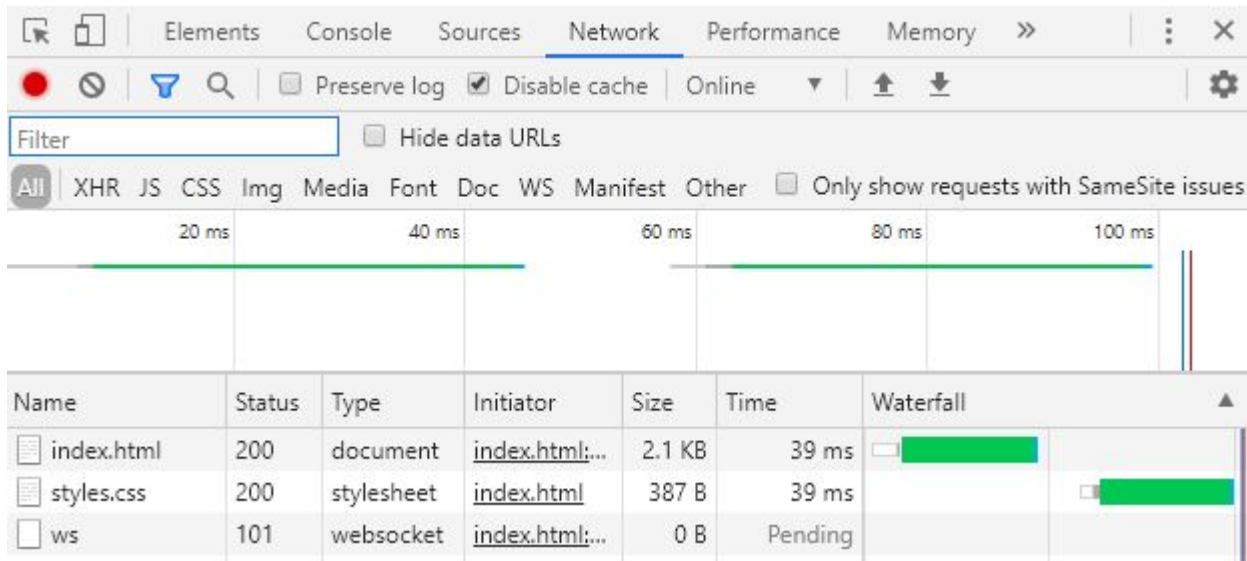
```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="css/styles.css">
</head>
```

`css/styles.css` мы написали сами, вместо того, что предложил Emmet.



# Developer Tools

Теперь в Developer Tools видно, что CSS подгружается:



Стили "применились"



# Developer Tools

**Важно:** Developer Tools (далее – DevTools), второй по важности инструмент для Frontend-разработчика, после редактора кода. Поэтому постоянно практикуйтесь в его использовании.



**JS**

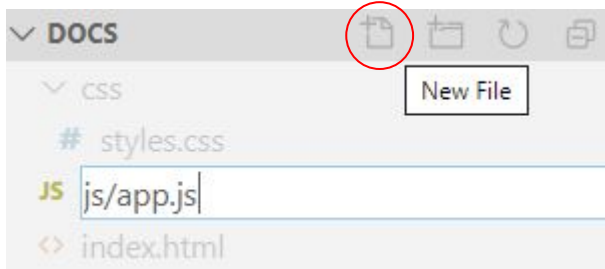




# JS

JavaScript (JS) – язык программирования, позволяющий взаимодействовать с загруженной в браузер веб-страницей (это не единственное применение JS, но для нас в рамках этого курса будет ключевым).

Поступим по аналогии с CSS – создадим файл `js/app.js` (чтобы он не попал внутрь каталога `css`, выделите либо `index.html`, либо кликните на свободной области боковой панели):



# JS

Введите в файл следующий код (что он значит и как работает, мы будем разбирать на следующей лекции):

JS app.js ×

js > JS app.js


```
1 alert('js worked!');
```



# JS

Естественно, этот код не будет работать, пока мы его не подключим в [index.html](#) (можете в этом удостовериться самостоятельно), поэтому наберите `script:src` и нажмите **Tab** (ещё одно сокращение Emmet):

```
<body>
  <h1>Hello, Alif Skills!</h1>
  script:src
</body>
</html>
```

An Emmet Abbreviation tooltip is shown, displaying a wrench icon, the text 'script:src', and the label 'Emmet Abbreviation'.

```
<body>
  <h1>Hello, Alif Skills!</h1>
  <script src="js/app.js"></script>
</body>
</html>
```

A red arrow points from the text 'Установите значение атрибута' to the 'src' attribute in the code snippet.

Установите значение атрибута



# Завершаем работу

Для завершения работы достаточно закрыть окно VS Code (Live Server автоматически прекратит свою работу). Браузер придётся закрыть самостоятельно.



# ИТОГИ



# ИТОГИ

В этой лекции мы обсудили достаточно много важных моментов:

1. Установка инструментов
2. Работу в VS Code
3. Основы HTML, CSS
4. Подключение ресурсов и обработку страницы браузером

В следующих лекциях мы будем опираться на то, что вы уже изучили в этой лекции (и не будем детально описывать процессы создания файлов, запуска Live Server и т.д.).



# ДОМАШНЕЕ ЗАДАНИЕ



# Орг.моменты

Практикум состоит из 8 обязательных занятий. Мы выкладываем новые занятия каждый понедельник в 14:00 (по Душанбе), кроме первой недели.

**Каждое воскресенье в 23:59 (по Душанбе) дедлайн** сдачи домашнего задания. Дедлайн – это предельный срок, до которого вы должны сдать ДЗ.

Если не успеете сдать в срок домашнее задание, тогда этот практикум будет для вас закончен и вы сможете зарегистрироваться на запуск следующего через несколько месяцев.

Все вопросы вы сможете задавать в [Телеграм канале](#).





# Д3: Web App

Создайте базовую структуру проекта, аналогичную той, что мы делали на лекции:

```
✓ css
  # styles.css
✓ js
  JS app.js
  <> index.html
```

В `body` расположите элемент `h1` с текстом `Web App`.

В таблице стилей для элемента `body` определите цвет фона серый (`grey`), а цвет текста – белый (`white`).

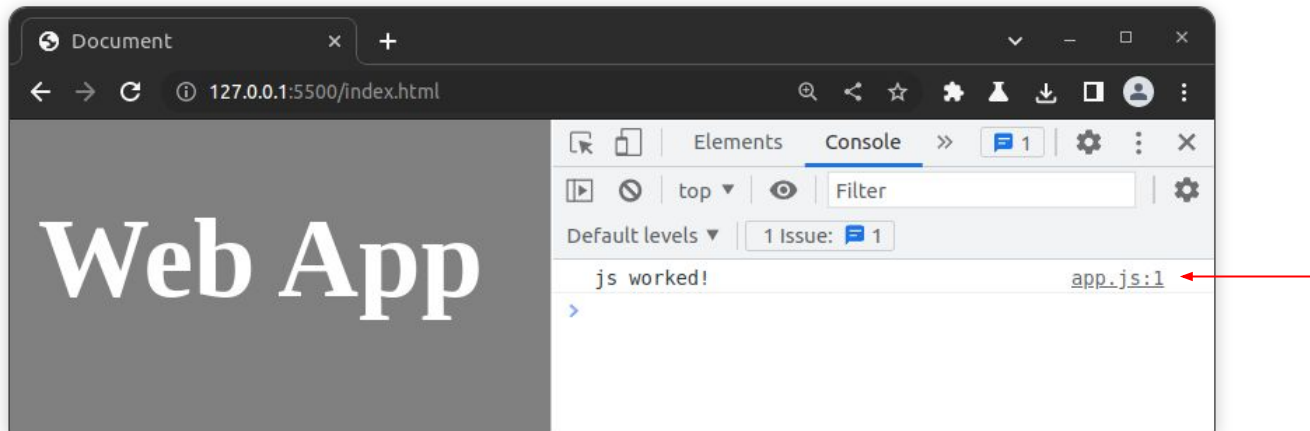
В файле `app.js` расположите следующую строку кода (остальное всё удалите):

```
console.log('js worked!');
```



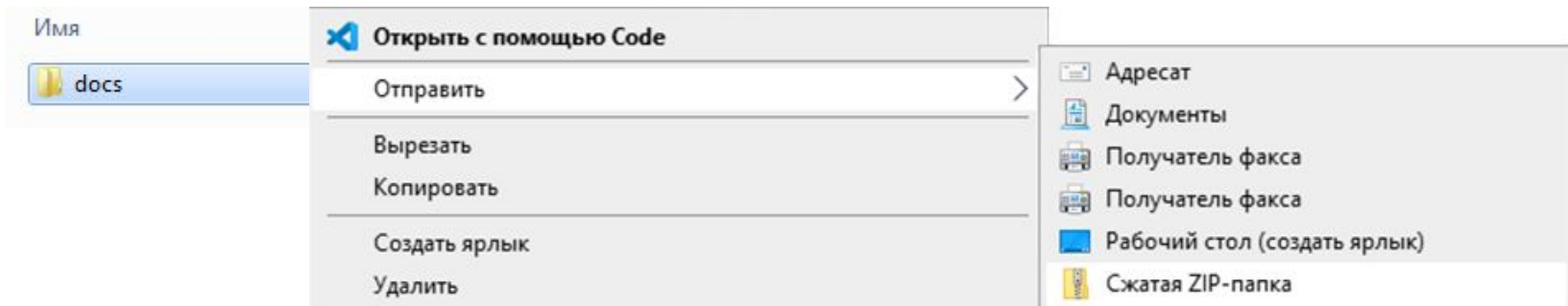
# Д3: Web App

Если вы всё сделаете правильно, то на вкладке **Console** DevTools увидите сообщение, которое разместили внутри [console.log](#):



# Как сдавать ДЗ

Вам нужно запаковать в zip-архив ваш каталог с проектом (не содержимое каталога, а сам каталог) – выделяете его и выбираете Отправить –> Сжатая ZIP-папка:



Полученный архив загружаете в личном кабинете пользователя.

**Важно:** учитывается только последняя отправленная попытка.



# Как сдавать ДЗ

В рамках первой ДЗ каталог должен называться `docs`.



# Успешная сдача

После того, как вы успешно выполните все требования задания, оно будет отмечено как зачтённое (пересдать его будет нельзя):

## Название ДЗ



Баллов: 1 Обязательна: Да Дедлайн: 01.01.2023 23:59:59

Задача сдана

✓ Последняя попытка

#30 Статус: **ПРИНЯТО** Отправлено: 01.01.2023 23:59:00 Проверено: 01.01.2023 23:59:01

```
START
Распаковываем архив - OK
Проверяем каталог docs - OK
Проверяем файл index.html (в каталоге проекта) - OK
Проверяем приложение
Пишем тест - OK
Запускаем тесты - OK
OK
```



# Успешная сдача

Только после сдачи всех обязательных задач к текущему занятию в срок (до дедлайна) вам будет доступны материалы следующего занятия (конечно же, как только это занятие будет опубликовано).



Спасибо за внимание

**alif skills**

2023г.

